Angular 2 – Assignments

[Assignments 4](#_Toc476430068)

[1. Quick Start Master 4](#_Toc476430069)

[2. Interpolation 4](#_Toc476430070)

[3. Using HTML template file in Angular components 4](#_Toc476430071)

[4. Multiple Components 4](#_Toc476430072)

[5. Built-in directives 4](#_Toc476430073)

[6. Property Binding 5](#_Toc476430074)

[7. Event Binding 5](#_Toc476430075)

[8. Two-way data binding 5](#_Toc476430076)

[9. Built-in pipe 5](#_Toc476430077)

[10. Custom pipe 6](#_Toc476430078)

[11. Component specific styles 6](#_Toc476430079)

[12. Passing data from outer component to inner component 6](#_Toc476430080)

[13. Communicating from inner component to outer component 6](#_Toc476430081)

[14. Lifecycle hooks 6](#_Toc476430082)

[15. Service 7](#_Toc476430083)

[16. Http Service 7](#_Toc476430084)

[17. Routing 7](#_Toc476430085)

[Development of a simple application – Bill Payments application 8](#_Toc476430086)

# Assignments

## 1. Quick Start Master

* Setup the local development environment using the Quick Start Master steps provided in <https://angular.io/docs/ts/latest/guide/setup.html>
* Check if you are able to run the application and see the output in browser.

**Note:**  You can use the Quick Start Master setup code as the **base code** to add the functionalities specified in the next set of assignments.

## 2. Interpolation

* Interpolation concepts [ i.e. Using of **{{ …}}** ]
  + Display in template view, a property defined in Component class
  + Display in template view, a combination of hard coded string and a property defined in Component class
  + Display in template view, a value returned by a method defined in Component class
  + Display in template view, a combination of hard coded string and a value returned by a method defined in Component class
  + Display in template view a heading (e.g. h3) and for an attribute of h3 (e.g. title), define the value of the attribute using a property defined in Component class

## 3. Using HTML template file in Angular components

* The Quick Master Setup code has defined the HTML elements in a template string (Refer Assignment 1).
* Instead of using a template string to define the HTML elements, define the **HTML template in a separate file** and specify it in the Component decorator and check if it works correctly.

## 4. Multiple Components

* The Quick Master Setup code has defined a single component - “App” component. (Refer Assignment 1).
* Enhance the application to create one more component, say “Users” component and include this component as an inner component of “App” component.
* The User Component can use an HTML template which defines a hard coded list of users in the HTML template itself.
* **Note:** Ensure the “App” module includes the newly created component correctly.

## 5. Built-in directives

* The “Multiple Components” code (Assignment 4) has defined the list of users as a hard coded list directly in HTML template.
* Instead of having the users hard coded in the HTML template, define the list of users (with id, name, location) as a JSON array in “Users” Component class.
* In the HTML template, use **ngFor** built-in directive to iterate through the list of users and display name and location of all the users in a HTML table (with column headings “Name” and “Location”).
* Next use **ngIf** built-in directive to decide if a table should be displayed or not based on whether there are some users in the JSON array or not [ **Note:** Try with both scenarios – with some users in the JSON array as well as with an empty JSON array].

## 6. Property Binding

* Property binding concepts [ i.e. Using of **[ …]** ]
  + Display in template view, a string property defined in Component class using the property binding syntax
  + Experiment with a boolean property defined in Component class using the property binding syntax
  + Try comparing Interpolation syntax and property binding syntax and analyze what are the differences [Try it for both string and boolean and derive your conclusion ].

## 7. Event Binding

* Event binding concepts [ i.e. Using of **(…)** ]
  + In HTML template, define a button and on click of the button, handle the event and update a counter property in the Component class and log it to the console to display the updated counter.
  + In HTML template, define a text box and on typing text in the text box, update the value of the text box in a property in the Component class and log it to the console to display the text updated in the property.

## 8. Two-way data binding

* Two-way data binding concepts [ i.e. Using of **ngModel, [(…)]**  ]
  + In Component class, define a string property and define a text box in HTML template and use two-way data binding to display an initial value in the text box and on changing the value in the text box, update back the property in Component class.
* **Note:** Ensure the “FormsModule” module of Angular is imported in “App” Module code.

## 9. Built-in pipe

* The “Built-in directives” code (Assignment 5) had used ngFor to iterate through the user array and displayed the user information as a table.
* Enhance the application to use built-in pipe like uppercase, lowercase to display the username in uppercase or lowercase.
* Try out built-in pipes like date, currency etc.

## 10. Custom pipe

* The “Built-in directives” code (Assignment 5) had used ngFor to iterate through the user array and displayed the user information as a table.
* Enhance the application to implement a search filter (based on username) and display only the list of users matching the specified filter. Implement your own custom pipe to implement the feature.

## 11. Component specific styles

* The “Multiple Components” code (Assignment 4) had defined two components, “App” component and “Users” component.
* Enhance the “App” component to display some hard-coded list of some items.
* Define CSS styles in different ways for “App” component and “Users” component
  + Only using global styles – i.e. styles.css in the top-level directory which is included in index.html.
  + Component specific style – affecting only “App” component elements
  + Component specific style – affecting only “Users” component elements
* **Note:** While specifying component specific styles, try specifying styles in two different ways
  + - as an array of strings
    - as a set of CSS files

## 12. Passing data from outer component to inner component

* The “Multiple Components” code (Assignment 4) had defined two components, “App” component and “Users” component.
* Enhance the “App” component to pass the name of the country to the “Users” component so that title of the “Users” component can display the country the users belong to.

## 13. Communicating from inner component to outer component

* The “Multiple Components” code (Assignment 4) had defined two components, “App” component and “Users” component.
* Enhance the “App” and “Users” component so that when a username among the list of users displayed is clicked, the “Users” component should communicate to the “App” component the user which was actually clicked.
* The “App” component should then display the selected user in the “App” component view.

## 14. Lifecycle hooks

* The “Multiple Components” code (Assignment 4) had defined two components, “App” component and “Users” component.
* Implement lifecycle hook methods, ngOnInit, ngOnChanges and ngOnDestroy.
* For ngOnInit and ngOnDestroy, check out the calling of these methods by the Angular framework by writing console log messages.
* For ngOnChanges, provide a text box in the “App” component for specifying the “country” name for the users being displayed in the “Users” component. Try inputting the country and pass it to “Users” component for display of the “country” name in the title of the “Users” component. As the country name is changed in the text box in the “App” component, check out the calling of ngOnChanges by Angular framework by writing the console log messages of the data received by “ngOnChanges” method.

## 15. Service

* The “Built-in directives” code (Assignment 5) had used ngFor to iterate through the user array and displayed the user information as a table.
* In Assignment 5, the users list was maintained in a JSON array in “Users” Component class.
* Modify the application so that the “Users” Component class obtains the user list from a “UserService”. The “UserService” can maintain the user list in a JSON array declared within the service class itself.
* **Note:** Ensure the “App” module includes the newly created service correctly.

## 16. Http Service

* The “Service” code (Assignment 15) had implemented a “UserService” to maintain the list in a JSON array declared within the service class itself.
* Modify the application so that the “UserService” instead of maintaining the list in a JSON array declared within the service class, uses Http to get the user list from a REST API enabled server.
* If you have the knowledge to write a REST API enabled server side application to provide the user list to the “UserService”, implement it **(OR)** otherwise, create a JavaScript file containing JSON array of user list and provide the path of the file as the URL path (**Note:** Using a local file is just a simulation of implementing the actual REST API enabled server code).
* **Note:** Ensure the “HttpModule” module of Angular is imported in “App” Module code.

## 17. Routing

* Implement client side routing using Angular router.
* Have two routes, “Users” and “Customers”. Accessing “Users” route should display the list of users and accessing “Customers” route should display the list of customers.
* Provide navigation link so that 2 links (Users and Customers) are displayed which the user of the application can click to access the corresponding route.
* While displaying the user list in a table, there should be a provision to click on individual user so that the detailed information for a selected user can be displayed using a “UserDetail” component. Similarly, implement for customer list also.

# Development of a simple application – Bill Payments application

* Write a single page application (SPA) using the Angular concepts learnt while implementing the above set of assignments. The functionality will be achieved using the following process.
  + The Admin will create the user list and the biller list. For each biller, the list of input fields that should be displayed when the bill payment is made should also be obtained and stored in the persistent medium. There will also be a provision within the admin module itself to simulate the generation of bills for each user.
  + As part of the User functionality, there should be a provision for a user to register with a set of billers. The users will be able to see the list of bills that require payment and they will make the payment. The user should be able to view the list of payments he / she has made. (Need not implement real payment service only a simulation is fine).

**Note:** The main requirement is that the client side portion of the application should be developed using Angular 2. For the various persistent data that is required for the application, depending upon the background of the developer implementing the application, following possibilities exists and you are free to choose any one of the following option.

* Implement a REST API based server to serve the required data. The server portion of the code can be implemented in JavaJ2EE (OR) .Net (OR) Node.js/Express.js (OR) any other server implementation

**(OR)**

* Have a database like sqlite3 or mongodb or any other database (having a JavaScript module in npm) to maintain the persistent data

**(OR)**

* Have a JavaScript based local storage to maintain the persistent data

**(OR)**

* At least have a local JSON object in memory to serve the required data. This may not persist the data across application instances. This should be the least preferred option.